컴퓨터학부 20162506 전연선

I. 개요

GNU make 유틸리티는 여러 프로그램의 그룹을 유지하기 위해 사용된다. 큰 프로젝트를 할수록 여러 개의 모듈로 나누어 개발을 하게 되는데, 이 중 하나의 모듈(파일)이 바뀌면 관계를 가지고 있는 다른 여러 파일들도 새롭게 컴파일을 통해 갱신해야 한다. 컴파일을 한번에 한 파일씩 하게 된다면 에러가 발생할 수 있어 비효율적이다. 이러한 상황에서 효율적으로 사용할 수 있는 기능이 make이다. 기본적으로 make는 Makefile의 이름을 가진 파일을 자동으로 인식하여 처리하는데, 여러 옵션을 추가합으로써 다양한 설정을 적용할 수 있다.

II. 설계

가) 목표 설정

make 유틸리티에서 제공하는 기능을 구현해 ssu_make의 규칙을 만든다. Makefile(또는 사용자가 '-f' 옵션을 통해 지정한 다른 이름의 Makefile)을 불러와 정의되어 있는 타겟(Target), 종속 항목(Dependency), 명령(Command)를 분류하고 규칙 정의 부분의 정보를 저장한다. 매크로가 정의 되어있는 경우에는 매크로가 참조된 곳은 매크로 값으로 바꿔서 인식할 수 있도록 한다. 또한 기존 make 유틸리티에 있는 일부 옵션도 구현함으로써 작업 디렉터리 변경, 사용법 출력, 의존성 그래프 출력 등의 기능을 할 수 있도록 한다.

나) 구조 설계

main 함수 :

① OPTION() : 메인함수 인자들을 getopt()함수를 이용하여 옵션들을 처리한다.

② openFile() : 전역변수로 선언되어 있는 Makefile(문자열 포인터 변수 "filename")을 open()함수를 통해 파일을 연다.

③ checkMkfile() : 성공적으로 연 Makefile의 규칙 검사를 실행한다.

④ categorize() : 규칙 검사를 마친 후 오류가 없는 경우 Makefile의 타겟, 종속 항목, 명령, 매크로 등을 분류해 구조체에 저장한다. 구조체는 ssu_make_1.h헤더파일에 struct makefile, struct macro_value, struct manage가 선언 되어있다. 분류 하는 과정에서 매크로가 참조된 경우 해당 매크로 값(value)으로 바꾸어서 구조체에 저장하도록 한다.

⑤ getTARGET() : main함수 인자로 받은 target과 macro 등을 처리한다. target이 없는 경우, 가장 상위에 있는 target을 기본값으로 처리한다.

i. SSUmake() : 사용자로부터 전달 받은 target을 인자로 함수를 호출하여 해당 target에 대하여 실행한다. make의 규칙에 맞게 타겟의 종속 구조를 파악한다. 종속 구조는 종속항목을 인자로 하여 재귀호출을 통해 파악할 수 있다. 모든 종속 항목에 대해 호출

이 끝난 후, 명령을 실행하여 사용자가 입력한 target을 처리한다.

⑤ printM() / printG() : 기존 OPTION()함수에서 옵션 m, 옵션 t가 설정된 경우 명령을 처리하지 않고 매크로와 의존성 그래프를 출력하도록 한다. 매크로 리스트의 경우 struct macro_value구조체를, 의존성 그래프의 경우 struct Makefile 구조체를 이용하여 출력할 수 있다.

III. 구현

가) int main(int argc, char* argv[])

① 매개변수 argc, argv[]:

i. argc : 실행 창에서 입력한 명령행 옵션의 개수

ii. argv : 실행 창에서 입력된 명령행 옵션의 문자열

② 기능 : 차례대로 OPTION(), openFile(), getTARGET(), categorize(), SSUmake() 등을 실행해 ssu_make를 실행한다.

나) void OPTION(int argc, char* argv[])

① 매개변수 argc, argv[] :

i. argc : 명령행 옵션의 개수 만큼 for()문을 작동해서 입력된 모든 옵션을 처리한다.

ii. argv[] : getopt() 함수를 이용하여 '-'뒤에 오는 옵션을 구분한다.

② 기능 : 입력된 모든 옵션을 받을 때까지 while()문을 통해 실행한다.
'-c' 옵션인 경우, 전역 변수로 선언되어 있던 'filename'을 재설정하고, '-c'옵션인 경우, 작업 디렉터리를 변경한다. chdir()함수를 이용하여 함수의 인자로는 사용자가 실행창에서 입력한 경로를 입력한다. '-s'옵션은 전역변수로 선언된 s_opt 값을 1로 바꾸어주어 ssu_make의 출력을 제한하도록 한다. '-h'옵션이 입력된 경우, PrintUsage()함수를 호출하여 옵션을 출력하여 바로 종료한다 (exit(0)). '-m'과 '-t'의 옵션이 설정 된 경우 전역변수로 선언되어 있던 각각의 변수를 1로 바꾸어 main()함수에서 진행하는 과정에서 옵션에 해당하는 그래프를 출력한다.

다) void openFile()

① 기능 : 전역변수 'filename'을 이름으로 가진 파일을 open()함수를 통해 열도록 한다. 이 때 에러가 발생할 경우, 에러 메시지를 출력한 후 종료한다(exit(1)).

라) void getTARGET(int argc, char *argv[], char (*target)[WORD], macro_value *Macro)

① 매개변수 argc, argv, target, Macro

i. argc : 명령행 옵션의 개수만큼 for()문을 작동 시켜 사용자가 입력한 모든 target이나 macro 지정값을 받는다. (i=1 ~ argc-1)

ii. argv : 명령행 옵션을 한 글자씩 읽으면서 argv[i]가 NULL문자를 만날 때까지 while()문을 사용하여 char형 배열에 저장한다. 입력 받은 글자에 '='이 포함된 경우 매크로 지정하는 경우이므로 Macro 구조체에 정보를 저장한다. '='이 포함되지 않고 옵션 설정이 아닌 경우 target을 입력한 것이므로 target에 저장하여 main()함수에서 ssu_make의 명령을 실행할 때 입력한 target 값을 인자로 전달한다.

iii. target : target이 입력된 경우 이 문자열 변수에 저장한다. target은 총 5개까지 받을 수 있고, 한 target당 최대 길이는 50byte이다.

iv. Macro : 사용자가 새 매크로를 지정하거나 기존 매크로를 변경한 경우 매크로가 선언되어 있는 macro_value 구조체에 변경된 정보를 저장한다. 기존 매크로를 변경하는 경우에는 해당되는 매크로의 value(Macro[i].value) 값만을 변경하고, 새로 지정하는 경우 매크로 구조체에 macro와 value 값 모두 저장하도록 한다.

② 기능 : 사용자로부터 target과 새로 지정하거나 변경하고자 하는 매크로를 입력 받는다. 입력받은 target 값은 main() 함수의 char형 배열 포인터(usr_tar[5][50]에 저장되어 SSUmake()를 실행할 때 인자 중 하나로 전달된다.

마) void categorize(makefile *info, macro_value *macroInfo)

① 매개 변수 info, macroInfo

i. info : 구조체에 target과 target에 해당하는 dependency(0~n번째 종속 항목까지)를 저장하고, command(0~n번째 명령)를 저장한다. target을 새로 저장할 때마다 전역변수로 선언된 manage ssu의 멤버 중 info_num의 값을 증가시켜 target의 개수를 저장한다.

ii. macroInfo : 구조체에 Makefile에 정의되어 있는 macro들을 저장한다. 저장할 때마다 전역변수로 선언된 manage ssu의 멤버중 macro_num의 값을 증가시켜 macro의 개수를 저장한다.

② 기능 : Makefile을 read() 함수를 사용하여 1byte씩 읽는다. 개행문자 ('₩n')가 올 때까지 char형 배열 buf에 저장한 후, 해당 buf를 명령어, target, dependency, 주석 등으로 구분한다.

buf의 0번째 자리에 '₩t'이 저장되어 있는 경우 명령으로 처리를 하고, '#'이 오는 경우는 주석으로 처리한다. 명령은 buf 전체를 info구조체의 멤버 중 command에 저장한다. target과 dependency는 buf를 공백 기준으로 나누어 나누어진 각각 문자열의 주소를 포인터배열에 저장한다.

저장된 문자열 중 ':'이 있는 경우 ':'를 기준으로 왼쪽에 있는 문자열은 info구조체의 멤버 target에, 오른쪽에 있는 문자열은 info구조체의 멤버 dependency에 저장한다. 문자열 중 '='이 오는 경우 macroInfo에 정보를 저장한다.

바) int SSUmake(makefile *Info, macro_value *macroInfo, char *usr_tar, int prev)

① 매개 변수 Info, macroInfo, usr_tar, prev

i. Info : 구조체에 Makefile의 target, dependency, command가 저장되어 있다.

ii. macroInfo : 구조체에 Makefile의 매크로와 매크로 값(value)이 저장되어 있다.

iii. usr_tar : 사용자가 입력한 target 값을 인자로 SSUmake를 호출한다. usr_tar값이 NULL값인 경우 제일 위에 저장되어 있는 target을 기본값으로 인자를 준다.

iv. prev : SSUmake()함수를 재귀적으로 호출할 때, 호출하는 함수의 target 순서가 저장되어 있는 값을 인자로 전달한다. 인자로 전달 된 prev는 호출 된 함수에서 의존성에 순환성이 존재하는 경우에 출력용으로 쓰인다.

② 기능 : 사용자로부터 받은 target에 대하여 ssu_make를 실행해 make 유틸리티와 같이 원하는 소스코드만 컴파일이 되도록 할 수 있다. 인자로 받은 usr_tar을 구조체에 저장된 타겟을과 비교한다. strcmp()함수를 통해 0이 반환된 경우 해당 target의 종속 항목들을 찾는다. 그리고 각 종속 항목들을 차례대로 SSUmake의 usr_tar 인자 값으로 넣어서 재귀 호출을 한다. 재귀 호출을 반복하여 각 종속 항목이 타겟에 없을 때까지 찾는다. 더 이상 타겟에 없을 경우 -1을 리턴하여 호출한 함수로 돌아간다. 호출된 함수에서 종속 항목에 대하여 모두 재귀호출이 끝난 경우 target에 해당하는 명령을 system()함수를 통해 실행한다.

재귀 호출을 반복할 때, 종속항목의 순환성에 주의해야 한다. 노드들의 깊이 우선 탐색과 같은 방식으로 생각하면서 전역 변수로 int형 배열 circ를 선언하였다. circ의 배열번호는 target의 순서와 같다. 새로운 함수를 호출 하면 circ[i]의 값을 1로 저장한다. 그리고 호출된 함수가 종료되면 해당 값은 다시 0으로 바꾼다. 이 과정에서 circ[i]의 값을 1로 저장하기 전 해당 칸에 이미 1이 저장되어 있는 경우, 중복이 존재한다는 의미이므로 순환성이 생긴다. 순환성이 생기면 순환을 의미하는 문장을 출력한 후 -1을 리턴하여 호출 전 함수로 돌아가도록 한다.

사) void PrintUsage()

① 기능 : 옵션 '-h'가 설정 된 경우 ssu_make의 옵션 사용법을 출력하는 함수이다.

아) void printMacro(macro_value *macroInfo)

① 기능 : macroInfo를 매개 변수로 해서 0번째부터 n번째(구조체 manage ssu의 macro_num) 매크로와 매크로 값(value)을 출력한다.

자) void printGraph(makefile *Info)

① 기능 : 첫번째 target부터 불러와 해당 target의 종속항목을 하위항목으로 출력하고, 각각의 종속항목의 이름을 가진 target들을 재귀적으로 호출하여 순환성이 생기거나 해당되는 target이 없을 때까지 반복한다. SSUmake()와 비슷한 방식이지만, printGraph에서는 command를 실행하는 것이 아니라 target을 출력하는 것으로 볼 수 있다.

②

IV. 테스트 및 결과

가) [OPTION]

① -f filename : ssu_make에서 실행시킬 Makefile을 filename으로 변경

```
junys@junys-VirtualBox:~/Phw$ mv Makefile ssu_makefile
junys@junys-VirtualBox:~/Phw$ ./ssu_make -f ssu_makefile
gcc -c test_code1.c
gcc -o test_make1  test_code1.o
```

Makefile을 ssu_makefile로 변경 후 실행

i. -f 옵션이 없는 경우 기본 값으로 Makefile을 사용

```
1 CC = gcc
2 USER = OSLAB
3 WELCOME = "WELCOME MESSAGE"
4 OS ?= LINUX
5
6 test_make1 : test_code1.o
7     $(CC) -o test_make1  test_code1.o
8 test_code1.o : test_code1.c
9     $(CC) -c test_code1.c
10
```

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make
gcc -c test_code1.c
gcc -o test_make1  test_code1.o
```

Makefile:

ii. filename을 이름으로 하는 파일이 없을 경우 에러 출력

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -f abcdefg
make: abcdefg: No such file or directory
```

② -c directory : 작업디렉터리를 현재 위치에서 directory로 변경

junys@junys-VirtualBox: ~/bhw

```
1 ssu_prog : p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o
2     gcc -Wall -W  p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o -o ssu_prog
3
4 p51_ssu_main.o : p51_ssu_main.c
5     gcc -Wall -W -c p51_ssu_main.c
6
7 p51_ssu_os.o : p51_ssu_os.c
8     gcc -Wall -W -c p51_ssu_os.c
9
10 p51_ssu_esp.o : p51_ssu_esp.c
11     gcc -Wall -W -c p51_ssu_esp.c
12
13 p51_ssu_lsp.o : p51_ssu_lsp.c
14     gcc -Wall -W -c p51_ssu_lsp.c
```

~/bhw에 있는 Makefile:

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -c ../bhw
gcc -Wall -W -c p51_ssu_main.c
gcc -Wall -W -c p51_ssu_os.c
gcc -Wall -W -c p51_ssu_esp.c
gcc -Wall -W -c p51_ssu_lsp.c
gcc -Wall -W  p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o -o ssu_prog
```

③ -s : command를 출력하지 않음

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -s
junys@junys-VirtualBox:~/Phw$
```

④ -h : 사용법 출력

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -h
Usage : ssu_make [Target] [Option] [Macro]
Option :
 -f <file>              Use <file> as a makefile.
 -c <directory> Change to directory <directory> before reading the makefiles.
 -s                             Do not print the commands as the are executed.
 -h                             print usage
 -m                             print macro list
 -t                             print tree
```

⑤ -m : ssu_make 실행 시 입력한 매크로 출력

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -m
--------------------------macro list-----------------------------
CC -> gcc
USER -> OSLAB
WELCOME -> "WELCOME
OS -> LINUX
```

나) [Target]

① Makefile에서 실행 될 target 지정

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -c ../bhw p51_ssu_main.o
gcc -Wall -W -c p51_ssu_main.c
```

② target이 없는 경우 가장 위에 있는 target을 기본 값으로 사용

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -c ../bhw
gcc -Wall -W -c p51_ssu_main.c
gcc -Wall -W -c p51_ssu_os.c
gcc -Wall -W -c p51_ssu_esp.c
gcc -Wall -W -c p51_ssu_lsp.c
gcc -Wall -W  p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o -o ssu_prog
```

다) [MACRO]

① Makefile에서 사용할 매크로 지정

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make LANG=KO -m
--------------------------macro list-----------------------------
CC -> gcc
USER -> OSLAB
WELCOME -> "WELCOME
OS -> LINUX
LANG -> KO
```

라) Makefile 규칙

① 규칙과 다른 경우 오류가 있는 라인 번호를 출력하고 종료

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make
Makefile:5: *** missing separator. Stop.
```

② 긴 명령어 '₩'의 사용

```
junys@junys-VirtualBox: ~/Phw
 1 ssu_prog : p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o
 2     gcc -Wall -W  p51_ssu_main.o \
 3 p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o -o ssu_prog
 4
 5 p51_ssu_main.o : p51_ssu_main.c
 6     gcc -Wall -W -c p51_ssu_main.c
 7
 8 p51_ssu_os.o : p51_ssu_os.c
 9     gcc -Wall -W -c p51_ssu_os.c
10
11 p51_ssu_esp.o : p51_ssu_esp.c
12     gcc -Wall -W -c p51_ssu_esp.c
13
14 p51_ssu_lsp.o : p51_ssu_lsp.c
15     gcc -Wall -W -c p51_ssu_lsp.c
```

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -c ../bhw
gcc -Wall -W -c p51_ssu_main.c
gcc -Wall -W -c p51_ssu_os.c
gcc -Wall -W -c p51_ssu_esp.c
gcc -Wall -W -c p51_ssu_lsp.c
gcc -Wall -W  p51_ssu_main.o p51_ssu_os.o p51_ssu_esp.o p51_ssu_lsp.o -o ssu_prog
```

③ 한 개 이상의 target이 존재해야 함

```
junys@junys-VirtualBox: ~/Phw
 1 CC = gcc
 2 USER = OSLAB
 3 WELCOME = "WELCOME MESSAGE"
 4 OS ?= LINUX
 5
 6
 7 #test_make1 : test_code1.o
 8 #   $(CC) -o test_make1  test_code1.o
 9 #test_code1.o : test_code1.c
10 #   $(CC) -c test_code1.c
11
```

-> 모든 target을 주석 처리함(target이 없는 경우)

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make
make: *** No targets. Stop.
```

④ dependency가 현재 디렉터리나 target에 없을 경우 에러 출력

```
junys@junys-VirtualBox:~/Phw$ ./ssu_make clear
make: *** No rule to make target 'clear'. Stop.
```

⑤ 의존성 그래프에 순환이 생긴 경우 중첩된 target부터는 실행하지 않음

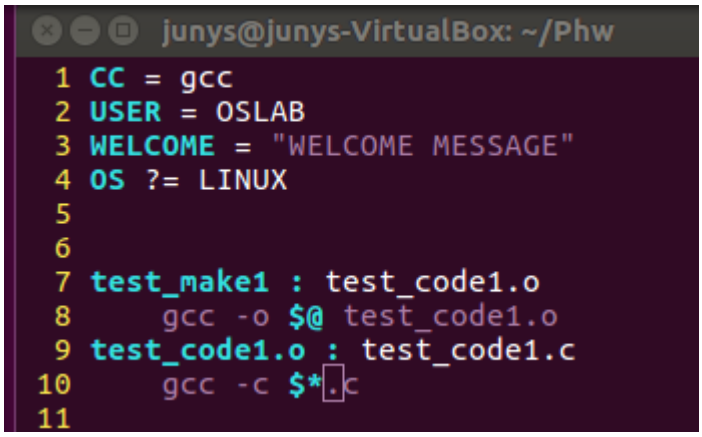
```
junys@junys-VirtualBox: ~/Phw
1 a : b
2     echo "aa"
3 b : c
4     echo "bb"
5 c : a b
6     echo "cc"
7 d : d
8     echo "dd"
```

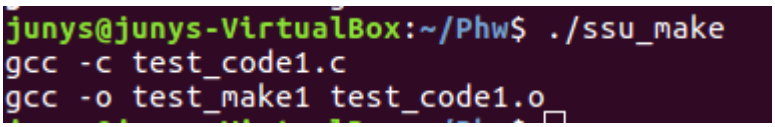
```
junys@junys-VirtualBox:~/Phw$ ./ssu_make -f echo
make : Circular c <- a dropped.
make : Circular c <- b dropped.
echo "cc"
cc
echo "bb"
bb
echo "aa"
aa
```

마) 내부 매크로

① $* : 확장자가 없는 target

② $@ : 현재 target





V. 소스 코드

가) ssu_make_1.h

```
#define BUFFER_SIZE 200
#define WORD 50

typedef struct makefile {
        char target[WORD];
        char dependency[WORD][WORD];
        char command[WORD][WORD];
} makefile;

typedef struct macro_value {
        char macro[WORD];
        char value[WORD];
} macro_value;

typedef struct manage {
        int info_num;
        int dep_num[WORD];
        int macro_num;
        int commnd_num[WORD];
} manage;
```

나) ssu_make.c

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#include "ssu_make_1.h"

//#define DEBUG_info
//#define DEBUG_macro
//#define DEBUG_COMMAND

int fd; //file descriptor of Makefile
char* filename = "Makefile";
int command = 0; //1 : command 실행 안됨
int s_opt = 0; //option -s
int printM = 0; //option -m
int printG = 0; //option -t
int level=0;
int circ[WORD] = {0}; //의존성 그래프 순환 확인

manage ssu; //Makefile의 target 수와 macro 관리

void openFile(); //open Makefile
int checkMkfile(); // check Makefile error
void categorize (makefile *info, macro_value *macroInfo);
int SSUmake(makefile *Info, macro_value *macroInfo, char *usr_tar, int prev); //execute Makefile
void getTARGET(int, char *argv[], char (*usr_tar)[], macro_value *usr_mac); //read target from user
void OPTION(int, char* argv[]);//process option
```

```
void PrintUsage();//option h
void printMacro(macro_value *macroInfo);//option m
void printGraph(makefile *Info);//optino t


int main(int argc, char* argv[])
{
	int error;
	int i;
	makefile Info[100] ={'₩0'}; //파일 각 target 별로 구조체에 의존성, command 저장
	macro_value Macro[100] = {'₩0'}; //매크로와 그 값 저장

	char usr_tar[5][WORD] = {'₩0'}; //사용자가 입력한 target 저장
	char circular[BUFFER_SIZE] = {'₩0'}; //의존성 그래프의 순환 확인
	OPTION(argc, argv);
	openFile();
	error = checkMkfile();
	if(error) exit(1); //end process

	//classify target, dependency, command, macro
	categorize(Info, Macro);
	//find target
	getTARGET(argc, argv, usr_tar, Macro);

	if(!command) {

		for(i=0; i<WORD; i++) {
```

```
                        if(usr_tar[i][0] != '₩0' && usr_tar[i][0] != '.') {
                                SSUmake(Info, Macro, usr_tar[i], 0);
                        }
                        else if(i==0 && ( usr_tar[i][0] == '.' || usr_tar[i][0] == '₩0')) {
                                SSUmake(Info, Macro, Info[0].target, 0); //target 없는 경우 기본 값으로
실행
                        }
                        else break;
                }

        }
        else {
                if(printM) printMacro(Macro);
                if(printG) printGraph(Info);
        }


        exit(0);
}

void openFile()
{
        //int error;
        if((fd = open(filename, O_RDONLY))<0 && !s_opt) { //Makefile이 없는 경우 에러 출력
                if(!s_opt)fprintf(stderr, "make: %s: No such file or directory₩n", filename);
                exit(1);
        }
```

```
}

int checkMkfile()
{
	off_t filesize = lseek(fd, (off_t)0, SEEK_END); //get size of fd

	char character; //파일로부터 1byte씩 character에 저장
	char line[BUFFER_SIZE] = {'₩0'}; // 파일 한 문장 line에 저장
	int lineNum = 0; //줄 번호
	int tar_com = 0;
	int error = 0; //error가 있는 경우 1

	int i = 0;
	int a = 0;

	lseek(fd, 0, SEEK_SET); //파일의 offset 시작점으로 변경
	while(read(fd, &character, 1) > 0) {
		line[i] = character;

		//공백으로 시작하는 경우
		if(line[0] == ' ') {
			error = 1;
			if(!s_opt)fprintf(stderr, " %s:%d: *** missing separator. Stop.₩n", filename, lineNum);
			break;
		}

		if(line[i] == ':') tar_com = 1;
```

```
		if(line[i] == '=') tar_com = 0;


		//긴 문장 처리 '₩'
		if(line[i] == '₩₩') {
			if((read(fd, &character, 1)) < 0) {
				error = 1;
				if(!s_opt)fprintf(stderr, "%s:%d: *** missing separator. Stop.₩n", filename,
lineNum);
				break;
			}
			if (character == '₩n');  //다음 문장 받음
			else { // '₩' 문자 뒤에 다른 문자가 오는 경우 에러
				error = 1;
				if(!s_opt)fprintf(stderr, "%s:%d: *** missing separator. Stop.₩n", filename,
lineNum);
				break;
			}
		}


		//한 문장이 끝난 경우
		else if(line[i] == '₩n') {
			lineNum++;


			if(line[0] == '	') { //'₩t'으로 시작한 경우
				if(!tar_com) { //command가 아닌 경우 error
					error = 1;
					printf("%s:%d: *** missing separator. Stop.₩n", filename, lineNum);
```

```
                        }
                }

                for(a = 0; a <= i; a++) line[a]='₩0'; //다음 문장 받기 위해 line 문자열 초기화
                i = 0;
        }
                else i++;
        }
        return error;
}

void categorize(makefile *info, macro_value *macroInfo)
{
        char buf[BUFFER_SIZE]; //파일 한 문장씩 line에 저장
        char token[WORD][WORD] = {'₩0'}; //공백 기준으로나눠진 단어 저장
        char *TOK[WORD] = {'₩0'}; //공백 기준으로 나눈 단어의 주소값 저장
        char character;
        char change_buf[WORD] = {'₩0'}; //매크로의 value값으로 바꾼 후 해당 문장
        char change_mac[WORD] = {'₩0'}; //바꿔야 할 매크로


        ssu.info_num = 0; //initialize number of targets
        ssu.macro_num = 0; //initialize number of macros

        int tok_num; // 한 문장에서 공백 기준으로 나누어진 단어 수
        int i = 0; //iteration
        int a = 0;
```

```
int letter; //한 글자씩 읽을 때 순서를 나타냄
int info_i = 0; //target 수
int tok = 0;
int dep_num = 0; //dependency number
int command = 0;
int com_num = 0; //한 target에서 command 순서
int com_continue = 0; //같은 target에 해당하는 command인지 아닌지 확인용
int continue_sent = 0; //긴 문장 처리 된 경우
int t=0;
int mac_start, mac_end; //매크로 시작부분과 끝 부분 배열 번호 저장
int no_macro = 0;
lseek(fd, 0, SEEK_SET); //파일의 offset 시작점으로 설정

while(read(fd, &character, 1) > 0) {
        buf[i] = character;

        if(buf[i] == '₩₩') {
                read(fd, &character, 1); //skip '₩n'
        }

        else if(buf[i] == '₩n') {

                ///Makefile 주석처리
                if(buf[0] == '#') {
                        for(a=0; a <= i; a++) buf[a] = '₩0';
                        continue;
                }
```

```
else if(buf[0] == '' || buf[0] == '₩t') { //'₩t'로 구분 된 command인 경우
    if(com_continue == 0) { //한 target에 대해서 첫번째 command인 경우
        com_num = 0;
    }

    for(a=0, i=1; i<strlen(buf); i++) {
        if(buf[i] == '$') { //매크로가 있는 경우
            no_macro = 0;
            mac_start = i++; //매크로 시작 부분 index 값 저장
            if(buf[i] == '(') { //Makefile에 정의되어 있는 매크로인 경우
                i++;
                while(buf[i] != ')') {
                    change_mac[a++] = buf[i++];
                }
                mac_end = i;
                for(a = 0; a < ssu.macro_num; a++) {
                    if(strcmp(change_mac, macroInfo[a].macro) == 0) { //해당 매크로가 정의되어 있는 지 확인
                        //change_buf에 매크로 값으로 바꿔서 저장
                        strncpy(change_buf, buf+1, mac_start-1);
                        strcat(change_buf, macroInfo[a].value);
                        strcat(change_buf,
```

buf+(mac_end +1));

info구조체에 저장

1].command[com_num++], change_buf);

chagne_buf

없는 현재의 목표 파일

info[ssu.info_num-1].target[i++];

target파일로 바꿔서 info구조체에 저장

```
                }
            }
            //해당 macro->value 값으로 바꿔서
            strcpy(info[ssu.info_num-
            for(a = 0; a < WORD; a++) { //intialize
                change_buf[a] = '₩0';
            }
        }
        else if(buf[i] == '*') { //내부매크로($*) : 확장자가
            mac_end = i;
            i = 0;
            while(info[ssu.info_num-1].target[i] != '.') {
                change_mac[a++] =
            }
            strncpy(change_buf, buf+1, mac_start-1);
            strcat(change_buf, change_mac);
            strcat(change_buf, buf+(mac_end+1));
            //$* 내부 매크로->확장자 없는 현재
            strcpy(info[ssu.info_num-
```

```
1].command[com_num++], change_buf);

                    for(a = 0; a < WORD; a++) { //initailize
change_buf
                        change_buf[a] = '₩0';
                    }
                }
                else if(buf[i] == '@') {;//내부 매크로($@) : 현재
목표 target
                    mac_end = i;
                    strcpy(change_mac, info[ssu.info_num-
1].target);

                    strncpy(change_buf, buf+1, mac_start-1);
                    strcat(change_buf, change_mac);
                    strcat(change_buf, buf+(mac_end+1));

                    //$@ 내부 매크로->현재 target 파일로
바꿔서 info구조체에 저장
                    strcpy(info[ssu.info_num-
1].command[com_num++], change_buf);

                    for(a = 0; a < WORD; a++) { //initailize
change_buf
                        change_buf[a] = '₩0';
                    }
                }
                else no_macro = 1;
                break;
```

```
                }
                else no_macro = 1;
            }

            if(no_macro) { //매크로 없는 경우
                strcpy(info[ssu.info_num-1].command[com_num++],
buf+1);
            }
            com_continue = 1;
            command = 1;
            ssu.commnd_num[ssu.info_num-1] = com_num;
        }
        else {
            //          command = 0;
            buf[i] = '₩0';//개행문자 대신 NULL 추가
            a = 0;
            TOK[a] = strtok(buf, " ");
            while(TOK[a] != NULL) { //" "기준으로 buf 잘라서 주소값 TOK[a]에
저장
                strcpy(token[a], TOK[a]);
                a++;
                TOK[a] = strtok(NULL, " ");
            }
            tok_num = a;

            for(tok = 0; tok < tok_num; tok++) {
                ////target과 dependency 정의
                if(token[tok][0] == ':') {
```

```
                com_continue = 0;
                t = 1; //':'앞 뒤 공백이 있는 경우
                dep_num = 0;
                strcpy(info[ssu.info_num].target, token[tok-1]);

        strcpy(info[ssu.info_num].dependency[dep_num++], token[++tok]);

                while(++tok != tok_num) {

        strcpy(info[ssu.info_num].dependency[dep_num++], token[tok]);
                }
                ssu.dep_num[ssu.info_num] = dep_num; //target에 해당하는 dependecy 개수 저
                ssu.info_num++; //Makefile 내 target 수++
                break;
        }

        //macro 정의장
        else if(token[tok][0] == '=' || token[tok][0] == '?') {
                t = 1; //'='?='앞 뒤 공백이 있는 경우
                strcpy(macroInfo[ssu.macro_num].macro, token[tok-1]);
                strcpy(macroInfo[ssu.macro_num].value, token[++tok]);
                ssu.macro_num++;
                break;
        }
```

```
                else t = 0; //token에 '='':'이 없는 경우
            }
            //token에 target,':',dependency모두 포함되어 있는 경우
            if (!t) {
                for(tok = 0; tok < tok_num; tok++) {
                    for(letter = 0; letter < sizeof(token[tok]); letter++)
{
                        //macro 정의
                        if(token[tok][letter] == '=' ||
token[tok][letter] == '?') {
                            for(a = 0; a < letter;
a++){ //macro 1byte씩 저장
        macroInfo[ssu.macro_num].macro[a] = token[tok][a];
                            }
                            for(a = letter+1; a <
sizeof(token[tok]); a++) { //value 값 1byte씩 저장
                                if(token[tok][letter] ==
'?')
        macroInfo[ssu.macro_num].value[a-(letter+1)] = token[tok][a+1];
                                else
        macroInfo[ssu.macro_num].value[a-(letter+1)] = token[tok][a];
                            }
                            ssu.macro_num++;
                            break;
                        }
```

```
                                                        //target과 dependency 정의
                                                        else if(token[tok][letter] == ':') {
                                                                com_continue = 0;
                                                                for(a = 0; a < letter;
a++){ //macro 1byte씩 저장

        info[ssu.info_num].target[a] = token[tok][a];

                                                                }
                                                                for (a = letter+1; a <
sizeof(token[tok]); a++) { //dependency 1byte씩 저장
                                                                        dep_num = 0;

        info[ssu.info_num].dependency[dep_num++][a-(letter+1)] = token[tok][a];
                                                                }
                                                                while(++tok != tok_num) {

        strcpy(info[ssu.info_num].dependency[dep_num++], token[tok]);
                                                                }
                                                                ssu.dep_num[ssu.info_num] =
dep_num;
                                                                ssu.info_num++;
                                                                break;
                                                        }
                                                        else; //아직 ':''='등 기호가 안나온 경우
                                                }
                                        }
                                }
                        }
```

```
                        for(i=0; i<WORD; i++) {
                                buf[i] = '₩0';
                        }
                        i = 0;
                }
                else i++;
        }

        if(ssu.info_num == 0) {
                if(!s_opt)printf("make: *** No targets. Stop.₩n");
                exit(0);
        }

#ifdef DEBUG_info
        printf("₩n");
        for(i=0; i < ssu.info_num; i++) {
                printf("#%d TARGET: %s, DEPENDENCY :",i, info[i].target);
                for(a=0; a < WORD; a++) {
                        printf("%s          ", info[i].dependency[a]);
                }
                printf("₩n");
        }
#endif

#ifdef DEBUG_macro
        for(i=0; i < ssu.macro_num; i++) {
                printf("#%d MACRO: %s = %s₩n",i, macroInfo[i].macro, macroInfo[i].value);
```

```
                }
#endif

#ifdef DEBUG_COMMAND
                for(i=0;i < ssu.info_num; i++) {
                        for(a=0; a < ssu.commnd_num[i]; a++ ) {
                                printf("#%d COMMAND : %s", i, info[i].command[a]);
                        }
                }
#endif
        }

        void getTARGET(int argc, char *argv[], char (*target)[WORD], macro_value *Macro)
        {
                int i;
                int tar_num = 0;
                int mac_num = 0;
                int a;
                //int mac_i = 0;
                int j = 0;
                char buf[WORD] = {'₩0'};
                char findmac[WORD] = {'₩0'};
                char usr_value[WORD] = {'₩0'};

                for(i = 1; i < argc; i++) {
                        j = 0;
                        if(argv[i][0] == '-'); //옵션이므로 다음 단어로 찾음
```

```
else {
        if (argv[i-1][0] != '-' && (argv[i-1][1] != 'f' || argv[i-1][1] != 'c')) {

            while(argv[i][j] != '₩0') {
            ////macro 사용자 정의////
                if(argv[i][j] == '=') {
                    strncpy(findmac, buf, j);

                    for(a = 0; a < ssu.macro_num; a++) {
                        if(strcmp(findmac, Macro[a].macro) == 0)
//Makefile에 macror가 이미 있는 경우
                            break;
                    }
                    mac_num = a;
                    if(a == ssu.macro_num) { //새로운 매크로 저장
                        ssu.macro_num++;
                        strcpy(Macro[mac_num].macro, findmac);
                    }
                    else { //기존의 macro value 값 초기화
                        for(a = 0; a <= WORD; a++) {
                            Macro[mac_num].value[a] = '₩0';
                        }
                    }
                    for(a = j+1; a < strlen(argv[i]); a++) { //새로운
value 저장
                        Macro[mac_num].value[a-(j+1)] =
argv[i][a];
                    }
```

```
                                        break;
                                }
                        ////argv[i][j]  저장////
                                else if(j+1 != strlen(argv[i])){
                                        buf[j] = argv[i][j];
                                        j++;
                                }

                        ////사용자가 입력한 target 저장
                                else {
                                        buf[j] = argv[i][j];
                                        strcpy(target[tar_num], buf);
                                        tar_num++;
                                        break;
                                }
                        }
                }
        }
}
int SSUmake(makefile *Info, macro_value *macroInfo, char *usr_tar, int prev)
{
        int tar = 0; //target 배열 번호
        int i, j, a;//iteration
        int new_tnum;
        //전역변수 circ[WORD] 사용
```

```
////Makefile에서 해당되는 target 찾기////
//target에 매크로 포함된 경우
while(strcmp(usr_tar, Info[tar].target) != 0) {
        tar++;
        //해당 target을 찾지 못한경우
        // 현재 디렉토리에 있는지 확인. 없는 경우 error

        if(tar == ssu.info_num) {

                if(level == 0 && !s_opt) printf("make: *** No rule to make target '%s'.
Stop.₩n", usr_tar);
                return -1;
        }
}

if(circ[tar] == 1) {
        if(!s_opt) printf("make : Circular %s <- %s dropped.₩n", Info[prev].target,
Info[tar].target);
        return -1;
}
circ[tar] = 1; //1 : 해당 target 방문

for(i = 0; i < ssu.dep_num[tar]; i++) {
        //dependency의 이름을 가진 target으로 SSUmake() 호출
        level++;
        new_tnum = SSUmake(Info, macroInfo, Info[tar].dependency[i],tar);
        if(new_tnum != -1) circ[new_tnum] = 0; //방문한 dependency 반환 (0)
}
```

```
		for(j = 0; j < ssu.commnd_num[tar]; j++) {
			if(!s_opt) printf("%s", Info[tar].command[j]);
			system(Info[tar].command[j]);
		}

		return tar;
}
void OPTION(int OptNum, char* option[])
{
		int c;

		while((c = getopt(OptNum, option, "f:c:shmt")) != -1) {
			switch(c) {
				case 'f': //option f
					filename	= optarg;
					break;

				case 'c': //option c
					chdir(optarg);
					//확인
					char buf[30];
					getcwd(buf, 30);
					break;

				case 's'://option s
					s_opt = 1;
					break;
```

```
                        case 'h': //option h
                                PrintUsage();
                                exit(0);
                                break;
                        case 'm'://option m
                                printM = 1;
                                command = 1;
                                break;
                        case 't'://option t
                                printG = 1;
                                command = 1;
                                break;
                }
        }
}

void PrintUsage()
{
        printf("Usage : ssu_make [Target] [Option] [Macro]₩n");
        printf("Option :₩n");
        printf(" -f <file>          Use <file> as a makefile.₩n");
        printf(" -c <directory>     Change to directory <directory> before reading the makefiles.₩n");
        printf(" -s                 Do not print the commands as the are executed.₩n");
        printf(" -h                 print usage₩n");
        printf(" -m                 print macro list₩n");
```

```
		printf(" -t                                    print tree₩n");
}

void printMacro(macro_value *macroInfo)
{
		int i;
		printf("-------------------------macro list----------------------------₩n");
		for(i = 0; i < ssu.macro_num; i++) {
			printf("%s -> %s₩n", macroInfo[i].macro, macroInfo[i].value);
		}
}

void printGraph(makefile *Info)
{
}
```